<https://www.youtube.com/watch?v=YfO28Ihehbk&t=12s>

<https://www.techbeamers.com/top-7-websites-for-python-programming-beginners/>

<https://www.techbeamers.com/python-multithreading-concepts/>

tutorials and quiz <https://www.techbeamers.com/lambda-function-usage-python/>

<https://www.tutorialspoint.com/python/index.htm>

* **List** is a collection which is ordered and changeable. Allows duplicate members.
* **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.
* **Set** is a collection which is unordered and unindexed.changeable. No duplicate members.
* **Dictionary** is a collection which is unordered, changeable and indexed. No duplicate members.

Dictionaries are indexed by keys.

A tuple is allowed to be hashed, for example, using it as a key for dictionaries

### **Q1. What is the difference between list and tuples in Python?**

|  |  |
| --- | --- |
| **LIST vs TUPLES** | |
| **LIST** | **TUPLES** |
| Lists are mutable i.e they can be edited. | Tuples are immutable (tuples are lists which can’t be edited). |
| Lists are slower than tuples. | Tuples are faster than list. |
| Syntax: list\_1 = [10, ‘Chelsea’, 20] | Syntax: tup\_1 = (10, ‘Chelsea’ , 20) |
|  | Tuple can be hashed for e.g as a key for dictionaries. |

list immutable meaning elements can be changed . can be diff types, add, remove values

lists are slower than tuples

in tuples , cannot change values,immutable

in set , doesn’t support duplicates and sequence

List = []

Tuples = ()

Dict = {}

sets

Like lists and dictionaries, tuples can be nested within each other.

Tuples can be created without the parentheses, by just separating the values with commas.  
Tuples are faster than lists, but they cannot be changed

Sets differ from lists in several ways, but share several list operations such as **len**.  
They are unordered, which means that they can't be indexed.  
They **cannot**contain duplicate elements.  
Due to the way they're stored, it's **faster**to check whether an item is part of a set, rather than part of a list.  
Instead of using **append**to add to a set, use **add**.  
The method **remove**removes a specific element from a set; **pop**removes an arbitrary element.

Sets can be combined using mathematical operations.  
The **union**operator **|** combines two sets to form a new one containing items in either.  
The **intersection**operator **&** gets items only in both.  
The **difference**operator **-** gets items in the first set but not in the second.  
The **symmetric difference** operator **^** gets items in either set, but not both.

### **key features of Python**

1. **interpreted** language (need not be compiled).coverts to intermediate language and then converts to machine language. Python is one of the most successful interpreted languages. When you write a Python script, it doesn’t need to get compiled before execution.
2. memory paramters need not be declared. It allocates and clears the memory by itself
3. **dynamically typed**, this means that you don’t need to state the types of variables when you declare them or anything like that. You can do things like x=111 and then x="I'm a string" without error
4. python is well suited to [**object orientated programming**](https://www.edureka.co/blog/python-class/) in that it allows the definition of classes along with composition and inheritance. Python does not have access specifiers (like C++’s public, private).
5. **Writing Python code is quick** but running it is often slower than compiled languages. Fortunately，Python allows the inclusion of C based extensions so bottlenecks can be optimized away and often are. The [numpy](https://www.edureka.co/blog/python-numpy-tutorial/) package is a good example of this, it’s really quite quick because a lot of the number crunching it does isn’t actually done by Python
6. Python finds **use in many spheres** – web applications, automation, scientific modeling, big data applications and many more. Alternatively, you can utilize it as a “glue” layer to work with other languages.
7. In Python, functions are first-class objects. This means that they can be assigned to variables, returned from other functions and passed into functions. Classes are also first class objects
8. [Python makes difficult things easy](https://xkcd.com/353/) so programmers can focus on overriding algorithms and structures rather than nitty-gritty low level details.

PEP stands for **Python Enhancement Proposal.**It is a set of rules that specify guidelines and best practices how to format/wriet Python code for maximum readability. consistency PEP 8, sometimes spelled PEP8 or PEP-8,

<https://learntocodewith.me/programming/python/python-2-vs-python-3/>

1. syntax , library diff

5/2 = 2.5 in 3.x but 5.0/2.0 = 2.5 in 2x

2. In Python 3, [text strings are Unicode by default](https://timothybramlett.com/Strings_Bytes_and_Unicode_in_Python_2_and_3.html). In Python 2, strings are stored as ASCII by default–you have to add a “u” if you want to store strings as Unicode in Python 2.x.

This is important because Unicode is more versatile than ASCII. Unicode strings can store foreign language letters, Roman letters and numerals, symbols, emojis, etc., offering you more choices.

### 3 THE TWO VERSIONS HAVE DIFFERENT PRINT STATEMENT SYNTAXES

**How can you do debugging in python ?**

We can do by using inbuilt module pdb. The [module pdb](https://docs.python.org/3/library/pdb.html) defines an interactive source code debugger for Python programs.

### **Q-92: List Down Some Of The PDB Commands For Debugging Python Programs?**

Here are a few PDB commands to start debugging Python code.

* Add breakpoint **(b)**
* Resume execution **(c)**
* Step by step debugging **(s)**
* Move to the next line **(n)**
* List source code **(l)**
* Print an expression **(p)**

**the sys** module’s **settrace()**

### **How is memory managed in Python?**

**Ans:**

1. Memory management in python is managed by **Python private heap space**. All Python objects and data structures are located in a private heap. The programmer does not have access to this private heap. The python interpreter takes care of this instead.
2. The allocation of heap space for Python objects is done by Python’s memory manager. The core API gives access to some tools for the programmer to code.
3. Python also has an inbuilt garbage collector, which recycles all the unused memory and so that it can be made available to the heap space.

# Data Structures

As we have seen in the previous lessons, Python supports the following data structures: **lists**, **dictionaries**, **tuples**, **sets**.  
  
**When to use a dictionary:**  
- When you need a logical association between a **key:value** pair.  
- When you need fast lookup for your data, based on a custom key.  
- When your data is being constantly modified. Remember, dictionaries are mutable.  
  
**When to use the other types:**  
- Use **lists**if you have a collection of data that does not need random access. Try to choose lists when you need a simple, iterable collection that is modified frequently.  
- Use a **set**if you need uniqueness for the elements.  
- Use **tuples**when your data cannot change.  
Many times, a **tuple**is used in combination with a **dictionary**, for example, a **tuple**might represent a key, because it's immutable.

# itertools

The module **itertools** is a standard library that contains several functions that are useful in functional programming.  
One type of function it produces is infinite iterators.  
The function **count** counts up infinitely from a value.  
The function **cycle** infinitely iterates through an iterable (for instance a list or string).  
The function **repeat**repeats an object, either infinitely or a specific number of times.

There are many functions in **itertools** that operate on iterables, in a similar way to **map**and **filter**.  
Some examples:  
**takewhile -** takes items from an iterable while a predicate function remains true;  
**chain -**combines several iterables into one long one;  
**accumulate -** returns a running total of values in an iterable.

There are also several combinatoric functions in **itertool**, such as **product**and **permutation**.  
These are used when you want to accomplish a task with all possible combinations of some items.

**List slices** provide a more advanced way of retrieving values from a list. Basic list slicing involves indexing a list with **two colon-separated integers**. This returns a new list containing all the values in the old list between the indices.

Slicing in Python is a mechanism to select a range of items from Sequence types like strings, list, tuple, etc. Python can also perform reverse indexing, i.e., in the backward direction, with the help of negative numbers

Like the arguments to **range**, the first index provided in a slice is included in the result, but the second isn't.

Slicing can also be done on tuples.

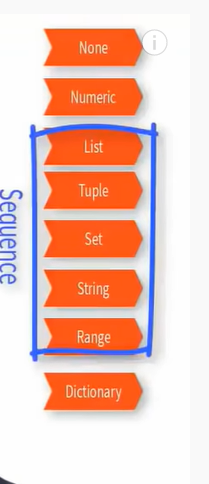
setting the env path

help topics

To create an empty set, you must use **set()**, as **{}** creates an empty dictionary.

<https://www.python-course.eu/python3_magic_methods.php>

<http://minhhh.github.io/posts/a-guide-to-pythons-magic-methods>



<https://www.sololearn.com/Course/Python/>

<https://courses.edx.org/register>

udacity

sams- teach python in 24 hours

plural sight

https://books.goalkicker.com/PythonBook/

<https://www.howtogeek.com/197947/how-to-install-python-on-windows/>

<http://minhhh.github.io/posts/a-guide-to-pythons-magic-methods>

[‎02/‎13/‎17 9:49 AM] Talakokkula, Sai Kishore (CONT):

https://www.a2hosting.com/kb/developer-corner/postgresql/connecting-to-postgresql-using-python

[‎02/‎13/‎17 9:50 AM] Talakokkula, Sai Kishore (CONT):

for sql server, https://blogs.msdn.microsoft.com/cdndevs/2015/03/11/python-and-data-sql-server-as-a-data-source-for-python-applications/

pip --proxy https://uxf002:Tirumala91@proxy.kdc.capitalone.com:8099 --trusted-host pypi.python.org install psycopg2

pip --proxy https://uxf002:Tirumala91@proxy.kdc.capitalone.com:8099 --trusted-host pypi.python.org install psycopg

pip --proxy https://uxf002:Tirumala91@proxy.kdc.capitalone.com:8099 --trusted-host pypi.python.org install kq

pip --proxy https://uxf002:Tirumala91@proxy.kdc.capitalone.com:8099 --trusted-host pypi.python.org install kafka-python

pip --proxy https://uxf002:Tirumala91@proxy.kdc.capitalone.com:8099 --trusted-host pypi.python.org install pyodbc

pip --proxy https://uxf002:Tirumala91@proxy.kdc.capitalone.com:8099 --trusted-host pypi.python.org install pymssql-2.1.3-cp36-cp36m-win\_amd64.whl

pip --proxy https://uxf002:Tirumala91@proxy.kdc.capitalone.com:8099 --trusted-host pypi.python.org install configparser

pip --proxy https://uxf002:Tirumala91@proxy.kdc.capitalone.com:8099 --trusted-host pypi.python.org install petl

pip --proxy https://uxf002:Tirumala91@proxy.kdc.capitalone.com:8099 --trusted-host pypi.python.org install nose

pip --proxy https://uxf002:Tirumala91@proxy.kdc.capitalone.com:8099 --trusted-host pypi.python.org install sqlalchemy

pip --proxy https://uxf002:Tirumala91@proxy.kdc.capitalone.com:8099 --trusted-host pypi.python.org install boto

pip --proxy https://uxf002:Tirumala91@proxy.kdc.capitalone.com:8099 --trusted-host pypi.python.org install bson

pip --proxy https:// ec2-user@proxy.kdc.capitalone.com:8099 --trusted-host pypi.python.org install future

[‎09/‎05/‎17 5:20 PM]  Kancham, Naveen:

pip --proxy http://ntv421:password@proxy.kdc.capitalone.com:8099 --trusted-host pypi.python.org install smart\_open -t C:\Work\WIP\08212017\pythonstuff

[‎09/‎05/‎17 5:24 PM]  Kancham, Naveen:

pip --proxy http://ntv421:password@proxy.kdc.capitalone.com:8099 --trusted-host pypi.python.org install pymssql-2.1.3-cp36-cp36m-win\_amd64.whl -t C:\Work\WIP\Demo08302017\ACBSDeploymentPackage

1.'hello'.capitalize()

2. def cls():

print('\n' \* 50)

3.

See email

Uppercase

Lists—linear representation of data-hetrogenous(int, string,float)

Fruits—apple,banana,cherry,date-----🡪homogenous

Fruits—apple,3,banana,2,cherry,5,date,1-----🡪heterogenous

Tuples=readonly data

Ex:days of week. We wont want 8 day week

Dictionary:unique identitifier

Ex:Aadhar, SSN.

Array:it depends on values something like index values

Dictionary : depends on keys

Functions: block of code within program used for reuse.

Modules:

P1 – f1, f2, f3

P2- f1,f2, p1, f1

Python

You can use excel also

Exceptions

EDxception Handling

Regular Expression\*

Classes – oops

APIE

Debugging in python

Logging

API->browser-🡪program(python)->database

CGI-common Gateway interface- Way in which python program is interacting with browser

Database Connectivity:

Pycharm

1)import random(number generator)

import sys

import os

2)print("Ravo")

3)#comment

4)‘’’

Comments

‘’’

5)name =”Derek”

print(name)

6) 5 main types in python

1.Numbers

2. Strings

3. Lists

4. Tuples

5. Dictionaries

7)Arithmetic operators

+,-,\*,/,%,\*\*(exponential calcaulations),//----7 of them

print(“5 +2=”,5+2)--- 5 +2= 7

print(“5 -2=”,5-2)---- 5 -2= 3

print(“5 \*2=”,5\*2)--- 5 \*2= 10

print(“5 /2=”,5/2)---- 5 /2= 2.5

print(“5 \*\*2=”,5\*\*2)--- 5 \*\*2= 25

print(“5 //2=”,5//2)---- 5 //2= 2

print(“5 %2=”,5%2)--- 5 %2= 1

print(“1+2-3\*2=”,1+2-3\*2)------ -3

print(“(1+2-3)\*2=”,(1+2-3)\*2)—0

8) quote="\"Always remember you are unique\""

multi\_line\_quote='''\"just

... like everyone else\"'''

print(“%s %s %s” %(‘I like the quote’,quote,multi\_line\_quote))

9)print('\n'\*5)

10)print("I don't like the ",end="")

print("newlines")

11)**lists**

a) grocery\_list=['Juice','Tomatoes','Potatoes','Bananas']

>>> print(grocery\_list)

['Juice', 'Tomatoes', 'Potatoes', 'Bananas']

b)print(“First\_item:” ,grocery\_list[0])

c)grocery\_list[0]="greenjuice"

>>> print("First\_item:" ,grocery\_list[0])

First\_item: greenjuice

d) >>> print(grocery\_list[1:3])

['Tomatoes', 'Potatoes']

e)

>>> other\_events=['Wash Car','Pickup Kids','Cash Check']

>>> to\_do\_list=[other\_events,grocery\_list]

>>> print(to\_do\_list)

[['Wash Car', 'Pickup Kids', 'Cash Check'], ['greenjuice', 'Tomatoes', 'Potatoes', 'Bananas']]

f) >>> print((to\_do\_list**[1][1]**))

Tomatoes

g) >>> grocery\_list.**append**('Onions')

>>> print(to\_do\_list)

[['Wash Car', 'Pickup Kids', 'Cash Check'], ['greenjuice', 'Tomatoes', 'Potatoes', 'Bananas', 'Onions']]

h) >>> grocery\_list.**insert**(1,'Pickle')

>>> print(to\_do\_list)

[['Wash Car', 'Pickup Kids', 'Cash Check'], ['greenjuice', 'Pickle', 'Tomatoes', 'Potatoes', 'Bananas', 'Onions']]

i) >>> grocery\_list.**remove**('Onions')

>>> print(to\_do\_list)

[['Wash Car', 'Pickup Kids', 'Cash Check'], ['greenjuice', 'Pickle', 'Tomatoes', 'Potatoes', 'Bananas']]

j) >>> grocery\_list.**sort**()

>>> print(to\_do\_list)

[['Wash Car', 'Pickup Kids', 'Cash Check'], ['Bananas', 'Pickle', 'Potatoes', 'Tomatoes', 'greenjuice']]

>>> grocery\_list.**reverse**()

>>> print(to\_do\_list)

[['Wash Car', 'Pickup Kids', 'Cash Check'], ['greenjuice', 'Tomatoes', 'Potatoes', 'Pickle', 'Bananas']]

k) >>> **del** grocery\_list[4]

>>> print(to\_do\_list)

[['Wash Car', 'Pickup Kids', 'Cash Check'], ['greenjuice', 'Tomatoes', 'Potatoes', 'Pickle']]

l) >>> to\_do\_list2=other\_events**+**grocery\_list

>>> print(to\_do\_list2)

['Wash Car', 'Pickup Kids', 'Cash Check', 'greenjuice', 'Tomatoes', 'Potatoes', 'Pickle']

m) >>> print(**len**(to\_do\_list2))

7

n) >>> print(**max**(to\_do\_list2))

greenjuice

o) >>> print(**min**(to\_do\_list2))

Cash Check

**Tuples**

Tuples are surrounded by () like a ball

pi\_tuple = (3,1,4,1,5,9)

**tuple to list**

new\_tuple=list(pi\_tuple)

**list to tuple**

new\_list=tuple(new\_tuple)

>>> **max**(new\_tuple)

9

>>> **min**(new\_tuple)

1

**Dictionaries**:

a)It consists of unique keys. Joins cannot happen. It uses curly braces{}.key Value Pair

b)>>> super\_villains={'Fiddler':'Isaac','Captain':'Leonard','Weather':'Mark','Mirror':'Sam','Pied':'Thomas'}

>>> print(super\_villains)

{'Weather': 'Mark', 'Captain': 'Leonard', 'Pied': 'Thomas', 'Mirror': 'Sam', 'Fiddler': 'Isaac'}

c)>>> **print(super\_villains['Captain'])**

Leonard

>>> print(super\_villains['Captain'])

Leonard

d) >>> **super\_villains['Pied']='hartley'**

>>> print(super\_villains)

{'Weather': 'Mark', 'Captain': 'Leonard', 'Pied': 'hartley', 'Mirror': 'Sam'}

e) >>> **print(super\_villains.get('Pied'))**

Hartley

f) >>> print(super\_villains.keys())

dict\_keys(['Weather', 'Captain', 'Pied', 'Mirror'])

g) >>> print(super\_villains.values())

dict\_values(['Mark', 'Leonard', 'hartley', 'Sam'])

**Conditions**

ifelse elif == != > >= <=

a)>>> age=21

>>> if age>16:

... print('you are old enough to drive')

... else:

... print('you are not old enough to drive')

...

you are old enough to drive

b) >>> if age>21:

... print('tractor')

... elif age>=16:

... print('car')

... else:

... print('no\_drive')

...

Car

**Logical Operators**

And,or,not

a)>>> if ((age>=1)and (age<=18)):

... print("bday")

... elif(age==21):

... print("major")

... elif not(age==30):

... print("middle")

... else:

... print("Heel")

...

Major

To know the python version

/usr/bin/python -V

/usr/bin/python3 –V

Ctrl + D

Addition,

Subtraction

Multiplication

Division

Floor

Modulo

Floats

Print statement

Escape characters

iNput

**Concatenation- same datatypes only**

Strings can also be **multiplied**by integers

**Type Conversion from 1 to other datatype**

**Assigning Variables**

Certain restrictions apply in regard to the characters that may be used in Python variable names. The only characters that are allowed are letters, numbers, and underscores. Also, they can't start with numbers.  
Not following these rules results in errors.

spaces are not allowed

Python is a case sensitive programming language. Thus, **Lastname**and **lastname**are two different variable names in Python.

**del** statement

**In-place operators** allow you to write code like 'x = x + 3' more concisely, as 'x += 3'.   
The same thing is possible with other operators such as **-, \*, /**and **%** as well.

Boolean True and False

**!=**

> and <, >= and <=

**if** expression:  
statements

**if** statements can be nested

Python uses **indentation**(white space at the beginning of a line) to delimit blocks of code. Other languages, such as C, use curly braces to accomplish this, but in Python indentation is mandatory; programs won't work without it. As you can see, the statements in the **if** should be indented.

Telusko video

01:04 introduction to python

05:14 Python setup

13:06 Getting started with python

27:02 variables in python

36:48 list in python

47:02 tuple / set in python

53:16 python set path in Windows and help

58:46 more on variables in python

01:07:31 data types in python

01:21:42 operators in python

01:32:24 number system conversion in python

01:40:17 IDLE previous command | clear screen

01:43:11 python BitWise operators

01:55:28 import math function in python

02:02:43 working with PyCharm | run | debug | trace | Py file

02:12:38 user input in python | command line input

02:23:57 if..elif..else statement in python

02:39:01 while loop in python

02:51:19 for loop in python

02:58:09 break.. continue..pass in python

03:07:57 printing patterns in python

03:15:49 Object Oriented Programming

03:22:29 Class and Object in Python

03:33:19 \_INIT\_METHOD

03:33:54 OPPs Concepts

03:55:30 Types of Methods in Python

04:06:46 Inner Class In Python

04:13:51 Inheritance

04:36:21 Operator Overloading

04:50:39 Method Overloading and Method Overriding

05:00:57 Exception Handling in Python

05:16:35 Multithreading

05:31:05 File Handling

05:43:06 Comments in Python

05:53:19 Linear Search in Python

06:16:42 Selection sort

06:24:05 MySQL Workbench Setup

06:31:59 Python Database Connection

06:41:00 GitHub

Map

<https://sites.google.com/view/learn-python-data-science/home?fbclid=IwAR2CeiVso-TG71P0-nm1WhXDWkSTNSAYrbrTWQ-hNFe1ixkQfqyBojcLaOQ>

### **Q23. What are python iterators?**

***Ans:***Iterators are objects which can be traversed though or iterated upon.

**Iteration** is simply the number of time/times a **loop** can be executed, while **loop** is the code which generate or causes expressions to be iterated **iteration** when the **loop** is executing. The above code is a for\_loop in which the execution of the statement "this is printed 10 times" will be iterated/repeated 10 times

generators : The way of implementing iterators are known as generators. It is a normal function except that it yields expression in the function.

Functions that return an iterable set of items are called generators

### **Q-95: Why And When Do You Use Generators In Python?**

A generator in Python is a function which returns an iterable object. We can iterate on the generator object using the **yield** keyword. But we can only do that once because their values don’t persist in memory, they get the values on the fly.

Generators give us the ability to hold the execution of a function or a step as long as we want to keep it. However, here are a few examples where it is beneficial to use generators.

* We can replace loops with generators for efficiently calculating results involving large data sets.
* Generators are useful when we don’t want all the results and wish to hold back for some time.
* Instead of using a callback function, we can replace it with a generator. We can write a loop inside the function doing the same thing as the callback and turns it into a generator.

### **Q-96: What Does The Yield Keyword Do In Python?**

The **yield** keyword can turn any function into a generator. It works like a standard return keyword. But it’ll always return a generator object. Also, a method can have multiple calls to the **yield** keyword.

See the example below.

def testgen(index):

weekdays = ['sun','mon','tue','wed','thu','fri','sat']

yield weekdays[index]

yield weekdays[index+1]

day = testgen(0)

print next(day), next(day)

#output: sun mon

### **Q29. How will you capitalize the first letter of string?**

***Ans:***In Python, the capitalize() method capitalizes the first letter of a string. If the string already consists of a capital letter at the beginning, then, it returns the original string.

**docstring in Python** A Python documentation string is known as docstring, it is a way of documenting Python functions, modules and classes. Docstrings are not actually comments, but, they are ***documentation strings***. These docstrings are within triple quotes. They are not assigned to any variable and therefore, at times, serve the purpose of comments as well.

### **Q27. What is pickling and unpickling?**

**Ans:** Pickle module accepts any Python object and converts it into a string representation and dumps it into a file by using dump function, this process is called pickling. While the process of retrieving original Python objects from the stored string representation is called unpickling.

### **Q33. What is the purpose of is, not and in operators?**

***Ans:***Operators are special functions. They take one or more values and produce a corresponding result.

is: returns true when 2 operands are true  (Example: “a” is ‘a’)

not: returns the inverse of the boolean value

in: checks if some element is present in some sequence

### **Q34. What is the usage of help() and dir() function in Python?**

**Ans:** Help() and dir() both functions are accessible from the Python interpreter and used for viewing a consolidated dump of built-in functions.

1. Help() function: The help() function is used to display the documentation string and also facilitates you to see the help related to modules, keywords, attributes, etc.
2. Dir() function: The dir() function is used to display the defined symbols.

### **Q35. Whenever Python exits, why isn’t all the memory de-allocated?**

**Ans:**

1. Whenever Python exits, especially those Python modules which are having circular references to other objects or the objects that are referenced from the global namespaces are not always de-allocated or freed.
2. It is impossible to de-allocate those portions of memory that are reserved by the C library.
3. On exit, because of having its own efficient clean up mechanism, Python would try to de-allocate/destroy every other object.

a

**24) What is the difference between Xrange and range?**

Xrange returns the xrange object while range returns the list, and uses the same memory and no matter what the range size is.

For the most part, xrange and range are the exact same in terms of functionality. They both provide a way to generate a list of integers for you to use, however you please. The only difference is that range returns a Python list object and x range returns an xrange object.

This means that xrange doesn’t actually generate a static list at run-time like range does. It creates the values as you need them with a special technique called yielding. This technique is used with a type of object known as generators. That means that if you have a really gigantic range you’d like to generate a list for, say one billion, xrange is the function to use.

This is especially true if you have a really memory sensitive system such as a cell phone that you are working with, as range will use as much memory as it can to create your array of integers, which can result in a Memory Error and crash your program. It’s a memory hungry beast.

If you want to write code that will run on both Python 2 and Python 3, use range() as the xrange funtion is deprecated in Python 3. range() is faster if iterating over the same sequence multiple times. xrange() has to reconstruct the integer object every time, but range() will have real integer objects.

**26) Mention what are the rules for local and global variables in Python?**

**Local variables**: If a variable is assigned a new value anywhere within the function's body, it's assumed to be local. Any variable declared inside a function is known as a local variable. This variable is present in the local space and not in the global space.

**Global variables**: Those variables that are only referenced inside a function are implicitly global. Variables declared outside a function or in global space are called global variables. These variables can be accessed by any function in the program.

Global variable – we can access anywhere in the program.

Local variable – we can access inside a function.

### **Q-83: What Is The Use Of Globals() Function In Python?**

The globals() function in Python returns the current global symbol table as a dictionary object.

Python maintains a symbol table to keep all necessary information about a program. This info includes the names of variables, methods, and classes used by the program.

All the information in this table remains in the global scope of the program and Python allows us to retrieve it using the globals() method.

Signature: globals()

Arguments: None

# Example: globals() function

x = 9

def fn():

y = 3

z = y + x

# Calling the globals() method

z = globals()['x'] = z

return z

# Test Code

ret = fn()

print(ret)

The output is:

12

**What is difference between a += b is not always a = a + b in python ?**

<https://stackoverflow.com/questions/6951792/a-b-not-the-same-as-a-a-b>

### **Q15. What is the difference between Python Arrays and lists?**

***Ans:***Arrays and lists, in Python, have the same way of storing data. But, arrays can hold only a single data type elements whereas lists can hold any data type elements.

### **Q16. What are functions in Python?**

***Ans:***A function is a block of code which is executed only when it is called. To define a [Python function](https://www.edureka.co/blog/python-functions), the **def** keyword is used.

### **Q17.What is \_\_init\_\_?**

***Ans:***\_\_init\_\_ is a method or constructor in [Python](https://www.edureka.co/blog/python-programming-language). This method is automatically called to allocate memory when a new object/ instance of a class is created. All classes have the \_\_init\_\_ method.

Here is an example of how to use it.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | class Employee:  def \_\_init\_\_(self, name, age,salary):  self.name = name  self.age = age  self.salary = 20000  E1 = Employee("XYZ", 23, 20000)  # E1 is the instance of class Employee.  #\_\_init\_\_ allocates memory for E1.  print(E1.name)  print(E1.age)  print(E1.salary) |

**Output:**

*[Powered by Edureka](https://www.edureka.co/mymock-interview-service" \t "_blank)*[[](https://www.edureka.co/mymock-interview-service)](https://www.edureka.co/mymock-interview-service" \t "_blank)

### [80% INTERVIEW REJECTIONS HAPPEN IN FIRST 90 SECONDS](https://www.edureka.co/mymock-interview-service" \t "_blank)

**[Take Python Mock Interview](https://www.edureka.co/mymock-interview-service" \t "_blank)**

* **[Get Interviewed by Industry Experts](https://www.edureka.co/mymock-interview-service" \t "_blank)**
* **[Personalized interview feedback](https://www.edureka.co/mymock-interview-service" \t "_blank)**

[BOOK A SLOT](https://www.edureka.co/mymock-interview-service" \t "_blank)

XYZ

23

20000

### **Q18.What is a lambda function?**

***Ans:***An anonymous function is known as a lambda function. This function can have any number of parameters but, can have just one statement.

### **Q19. What is self in Python?**

***Ans:***Self is an instance or an object of a class. In Python, this is explicitly included as the first parameter. However, this is not the case in Java where it’s optional.  It helps to differentiate between the methods and attributes of a class with local variables.

The self variable in the init method refers to the newly created object while in other methods, it refers to the object whose method was called.

### **Q20. How does break, continue and pass work?**

|  |  |
| --- | --- |
| Break | Allows loop termination of execution when some condition is met and the  control is transferred to the next statement.  It can be used  in for and while loops |
| Continue | Allows skipping some part of a loop when some specific condition is met  and the control is transferred to the beginning of the loop |
| Pass | Used when you need some block of code syntactically, but you want to skip  its execution. This is basically a null operation. Nothing happens when this is  executed. |

### **Q21.**What does [::-1} do?

***Ans:*** [::-1] is used to reverse the order of an array or a sequence.

### **Q22. How can you randomize the items of a list in place in Python?**

**Ans:** Consider the example shown below:

|  |  |
| --- | --- |
| 1  2  3  4 | from random import shuffle  x = ['Keep', 'The', 'Blue', 'Flag', 'Flying', 'High']  shuffle(x)  print(x) |

The output of the following code is as below.

['Flying', 'Keep', 'Blue', 'High', 'The', 'Flag']

### **What is enumerate() function in python ?**

#### 4. Use of Enums in Python

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | class Training:  Mindmajix, Online, Mindmajix = range(3)    print(MyName.Mindmajix)  print(MyName.Online)  print(MyName.Mindmajix) |

**Output:**

|  |  |
| --- | --- |
| 1  2  3 | 2  1  2 |

While using the iterators, sometimes we might have a use case to store the count of iterations. Python gets this task quite easy for us by giving a built-in method known as the enumerate().

The enumerate() function attaches a counter variable to an iterable and returns it as the “enumerated” object.

We can use this object directly in the “for” loops or transform it into a list of tuples by calling the list() method. It has the following signature:

enumerate(iterable, to\_begin=0)

Arguments:

iterable: array type object which enables iteration

to\_begin: the base index for the counter is to get started, its default value is 0

# Example - enumerate function

alist = ["apple","mango", "orange"]

astr = "banana"

# Let's set the enumerate objects

list\_obj = enumerate(alist)

str\_obj = enumerate(astr)

print("list\_obj type:", type(list\_obj))

print("str\_obj type:", type(str\_obj))

print(list(enumerate(alist)) )

# Move the starting index to two from zero

print(list(enumerate(astr, 2)))

The output is:

list\_obj type: <class 'enumerate'>

str\_obj type: <class 'enumerate'>

[(0, 'apple'), (1, 'mango'), (2, 'orange')]

[(2, 'b'), (3, 'a'), (4, 'n'), (5, 'a'), (6, 'n'), (7, 'a')]

### **Q37. How can the ternary operators be used in python?**

**Ans:** The Ternary operator is the operator that is used to show the conditional statements. This consists of the true or false values with a statement that has to be evaluated for it.

x, y = 25, 50 big = x if x < y else y

### **Q39. What does len() do?**

***Ans:***It is used to determine the length of a string, a list, an array, etc.

### **Q40. Explain split(), sub(), subn() methods of “re” module in Python.**

**Ans:** To modify the strings, Python’s “re” module is providing 3 methods. They are:

* split() – uses a regex pattern to “split” a given string into a list.

you can use split() function to split the string by comma or space etc.

* sub() – finds all substrings where the regex pattern matches and then replace them with a different string
* subn() – it is similar to sub() and also returns the new string along with the no. of replacements.

### **Q41. What are negative indexes and why are they used?**

**Ans:** The sequences in Python are indexed and it consists of the positive as well as negative numbers. The numbers that are positive uses ‘0’ that is uses as first index and ‘1’ as the second index and the process goes on like that.

In Python, we can access both arrays & lists using a positive or negative numbers (aka index). A negative index reads the list elements from the end counting in the backward direction. Check out from the example given below.

The index for the negative number starts from ‘-1’ that represents the last index in the sequence and ‘-2’ as the penultimate index and the sequence carries forward like the positive number.

The negative index is used to remove any new-line spaces from the string and allow the string to except the last character that is given as S[:-1]. The negative index is also used to show the index to represent the string in correct order.

**How are arguments passed by value or by reference?**

In python, Objects passed as arguments to functions are passed by reference; they are not being copied around. Thus, passing a large list as an argument does not involve copying all its members to a new location in memory. Note that even integers are objects. However, the distinction of by value and by reference present in some other programming languages often serves to distinguish whether the passed arguments can be actually changed by the called function and whether the calling function can see the changes. Passed objects of mutable types such as lists and dictionaries can be changed by the called function and the changes are visible to the calling function. Passed objects of immutable types such as integers and strings cannot be changed by the called function; the calling function can be certain that the called function will not change them

Neither the arguments are Pass by Value nor does Python supports Pass by reference. Instead, they are Pass by assignment.

The parameter which you pass is originally a reference to the object not the reference to a fixed memory location. But the reference is passed by value. Additionally, some data types like strings and tuples are immutable whereas others are mutable.

### **Q42. What are Python packages?**

***Ans:***Python packages are namespaces containing multiple modules.

To keep your programs manageable as they grow, you may want to break them up into several files. Python allows you to put multiple function definitions into a file and use them as a module. You can import these modules into other scripts and programs. These files must have a .py extension.

**Modules -- > packages 🡪 libraries**

### **Q44. What are the built-in types of python?**

***Ans:***Built-in types in Python are as follows –

* Integers
* Floating-point
* Complex numbers
* Strings
* Boolean
* Built-in functions

### **Q45. What advantages do NumPy arrays offer over (nested) Python lists?**

**Ans:**

1. Python’s lists are efficient general-purpose containers. They support (fairly) efficient insertion, deletion, appending, and concatenation, and Python’s list comprehensions make them easy to construct and manipulate.
2. They have certain limitations: they don’t support “vectorized” operations like elementwise addition and multiplication, and the fact that they can contain objects of differing types mean that Python must store type information for every element, and must execute type dispatching code when operating on each element.
3. [NumPy](https://www.edureka.co/blog/python-numpy-tutorial/) is not just more efficient; it is also more convenient. You get a lot of vector and matrix operations for free, which sometimes allow one to avoid unnecessary work. And they are also efficiently implemented.
4. NumPy array is faster and You get a lot built in with NumPy, FFTs, convolutions, fast searching, basic statistics, linear algebra, [histograms](https://www.edureka.co/blog/python-matplotlib-tutorial/#Histogram), etc.

### **Q46. How to add values to a python array?**

***Ans:***Elements can be added to an array using the **append()**, **extend()** and the **insert (i,x)** functions.

<https://www.edureka.co/mymock-interview-service>

### **Q49. What is the difference between deep and shallow copy?**

***Ans:*** Shallow copy is used when a new instance type gets created and it keeps the values that are copied in the new instance. Shallow copy is used to copy the reference pointers just like it copies the values. These references point to the original objects and the changes made in any member of the class will also affect the original copy of it. Shallow copy allows faster execution of the program and it depends on the size of the data that is used.

Deep copy is used to store the values that are already copied. Deep copy doesn’t copy the reference pointers to the objects. It makes the reference to an object and the new object that is pointed by some other object gets stored. The changes made in the original copy won’t affect any other copy that uses the object. Deep copy makes execution of the program slower due to making certain copies for each object that is been called.

**Ans.** We can either use a **“Shallow Copy”** or follow a **“Deep Copy**” approach.

#### **Shallow Copy Method.**

The content of an object (say dictionary) doesn’t get copied by value but by creating a new reference.

>>> a = {1: [1,2,3]}

>>> b = a.copy()

>>> a, b

({1: [1, 2, 3]}, {1: [1, 2, 3]})

>>> a[1].append(4)

>>> a, b

({1: [1, 2, 3, 4]}, {1: [1, 2, 3, 4]})

#### **Deep Copy Method.**

It copies all the contents by value.

>>> c = copy.deepcopy(a)

>>> a, c

({1: [1, 2, 3, 4]}, {1: [1, 2, 3, 4]})

>>> a[1].append(5)

>>> a, c

({1: [1, 2, 3, 4, 5]}, {1: [1, 2, 3, 4]})

### **Q50. How is Multithreading achieved in Python?**

**Ans:**

1. Python has a multi-threading package but if you want to multi-thread to speed your code up, then it’s usually not a good idea to use it.
2. Python has a construct called the Global Interpreter Lock (GIL). The GIL makes sure that only one of your ‘threads’ can execute at any one time. A thread acquires the GIL, does a little work, then passes the GIL onto the next thread.
3. This happens very quickly so to the human eye it may seem like your threads are executing in parallel, but they are really just taking turns using the same CPU core.
4. All this GIL passing adds overhead to execution. This means that if you want to make your code run faster then using the threading package often isn’t a good idea.

### **Q51. What is the process of compilation and linking in python?**

**Ans:** The compiling and linking allows the new extensions to be compiled properly without any error and the linking can be done only when it passes the compiled procedure. If the dynamic loading is used then it depends on the style that is being provided with the system. The python interpreter can be used to provide the dynamic loading of the configuration setup files and will rebuild the interpreter.

The steps that are required in this as:

1. Create a file with any name and in any language that is supported by the compiler of your system. For example file.c or file.cpp
2. Place this file in the Modules/ directory of the distribution which is getting used.
3. Add a line in the file Setup.local that is present in the Modules/ directory.
4. Run the file using spam file.o
5. After a successful run of this rebuild the interpreter by using the make command on the top-level directory.
6. If the file is changed then run rebuildMakefile by using the command as ‘make Makefile’.

### **Q52. What are Python libraries? Name a few of them.**

Python libraries are a collection of Python packages. Some of the majorly used python libraries are – [Numpy](https://www.edureka.co/blog/python-numpy-tutorial/), [Pandas](https://www.edureka.co/blog/python-pandas-tutorial/), [Matplotlib](https://www.edureka.co/blog/python-matplotlib-tutorial/), [Scikit-learn](https://www.edureka.co/blog/scikit-learn-machine-learning/) and many more.

### **Q55. Explain Inheritance in Python with an example.**

**Ans:** Inheritance allows One class to gain all the members(say attributes and methods) of another class. Inheritance provides code reusability, makes it easier to create and maintain an application. The class from which we are inheriting is called super-class and the class that is inherited is called a derived / child class.

They are different types of inheritance supported by Python:

1. Single Inheritance – where a derived class acquires the members of a single super class.
2. Multi-level inheritance – a derived class d1 in inherited from base class base1, and d2 are inherited from base2.
3. Hierarchical inheritance – from one base class you can inherit any number of child classes
4. Multiple inheritance – a derived class is inherited from more than one base class.

### **Q58. Does python support multiple inheritance?**

**Ans:**Multiple inheritance means that a class can be derived from more than one parent classes. Python does support multiple inheritance, unlike Java.

### **Q-63: What Is Composition In Python?**

The composition is also a type of inheritance in Python. It intends to inherit from the base class but a little differently, i.e., by using an instance variable of the base class acting as a member of the derived class.

See the below diagram.

To demonstrate composition, we need to instantiate other objects in the class and then make use of those instances.

class PC: # Base class

processor = "Xeon" # Common attribute

def \_\_init\_\_(self, processor, ram):

self.processor = processor

self.ram = ram

def set\_processor(self, new\_processor):

processor = new\_processor

def get\_PC(self):

return "%s cpu & %s ram" % (self.processor, self.ram)

class Tablet():

make = "Intel"

def \_\_init\_\_(self, processor, ram, make):

self.PC = PC(processor, ram) # Composition

self.make = make

def get\_Tablet(self):

return "Tablet with %s CPU & %s ram by %s" % (self.PC.processor, self.PC.ram, self.make)

if \_\_name\_\_ == "\_\_main\_\_":

tab = Tablet("i7", "16 GB", "Intel")

print(tab.get\_Tablet())

The output is:

Tablet with i7 CPU & 16 GB ram by Intel

### **Q59. What is Polymorphism in Python?**

**Ans:**Polymorphism means the ability to take multiple forms. So, for instance, if the parent class has a method named ABC then the child class also can have a method with the same name ABC having its own parameters and variables. Python allows polymorphism.

### **Q61. How do you do data abstraction in Python?**

**Ans:**Data Abstraction is providing only the required details and hiding the implementation from the world. It can be achieved in Python by using interfaces and abstract classes.

### **Q62 Does python make use of access specifiers?**

**Ans:**Python does not deprive access to an instance variable or function. Python lays down the concept of prefixing the name of the variable, function or method with a single or double underscore to imitate the behavior of protected and private access specifiers.

### **Q63. How to create an empty class in Python?**

**Ans:**An empty class is a class that does not have any code defined within its block. It can be created using the *pass*keyword. However, you can create objects of this class outside the class itself. IN PYTHON THE PASS command does nothing when its executed. it’s a null statement.

**For example-**

|  |  |
| --- | --- |
| 1  2  3  4  5 | class a:      pass  obj=a()  obj.name="xyz"  print("Name = ",obj.name) |

**Output:**

Name = xyz

### **Q64. What does an object() do?**

**Ans:**It returns a featureless object that is a base for all classes. Also, it does not take any parameters.

### **Q57. What is monkey patching in Python?**

**Ans:** In Python, the term monkey patch only refers to dynamic modifications of a class or module at run-time.

Consider the below example:

|  |  |
| --- | --- |
| 1  2  3  4 | # m.py  class MyClass:  def f(self):  print "f()" |

We can then run the monkey-patch testing like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | import m  def monkey\_f(self):  print "monkey\_f()"    m.MyClass.f = monkey\_f  obj = m.MyClass()  obj.f() |

The output will be as below:

monkey\_f()

Monkey patching is changing the behaviour of a function or object after it has already been defined. For example:

**import** datetime

datetime.datetime.now = **lambda**: datetime.datetime(2012, 12, 12)

Most of the time it's a pretty terrible idea - it is usually best if things act in a well-defined way. One reason to monkey patch would be in testing. The [mock](https://pypi.python.org/pypi/mock) package is very useful to this end.

### **Q66. Write a program in Python to produce Star triangle.**

|  |  |
| --- | --- |
| 1  2  3  4 | def pyfunc(r):      for x in range(r):          print(' '\*(r-x-1)+'\*'\*(2\*x+1))  pyfunc(9) |

**Output:**

\*

\*\*\*

\*\*\*\*\*

\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Fibonacci Series

Method 1

**def** fib(n):  
 **if** n == 1:  
 **return** 0  
 **elif** n == 2 :  
 **return** 1  
 **elif** n <= 0 :  
 **return "Incorrect Input"  
 else** :  
 **return** fib(n-1) + fib(n-2)  
  
**print**(fib(9))

*#0,1,1,2*

*Method 2*

fib\_array = [0,1]  
  
**def** fib(n):  
 **if** n <= 0 :  
 **print**(**"Incorrect Input"**)  
 **elif** n == 0 :  
 **return** 0  
 **elif** n <= len(fib\_array) :  
 **return** fib\_array[n-1]  
 **else**:  
 temp\_fib = fib(n-1) + fib(n-2)  
 fib\_array.append(temp\_fib)  
 **return** temp\_fib  
  
**print**(fib(9))

Method 3

**def** fib(n):  
 a = 0  
 b = 1  
 **if** n == 1:  
 **return** a  
 **elif** n == 2 :  
 **return** b  
 **else**:  
 **for** x **in** range(2,n):  
 c = a+b  
 a,b = b,c  
 **return** c  
  
**print**(fib(9))

take Fibonacci series number of 100 but print only numbers < 100

**def** fib(n):  
 a = 0  
 b = 1  
 **if** n == 1:  
 **print**(a)  
 **elif** n <= 0:  
 **print**(**'Incorrect Input'**)  
 **else**:  
 **print**(a)  
 **print**(b)  
 **for** x **in** range(2,n):  
 c = a+b  
 a,b = b,c  
 **if** c > 100:  
 **break  
 else** :  
 **print**(c) *#if c < 100 else print('greater than 100')*fib(100)

### **Write a program in Python to check if a number is prime**

**def** prime(n):  
 **if** n <= 1:  
 **print** (n ,**' < = 1 .Give Higher Number '**)  
 **elif** n == 2:  
 **print**(n, **"is a Prime Number"**)  
 **else**:  
 **for** i **in** range(2,n):  
 **if** n%i == 0 :  
 **print**(n , **" not Prime Number"**)  
 **break  
 else** :  
 **print**(n, **" prime number "**)  
  
prime(9)

print prime numbers till n python

**def** prime(n):  
 **if** n <= 1:  
 **print** (n ,**' < = 1 .Give Higher Number '**)  
 **elif** n == 2:  
 **print**(n, **"is a Prime Number"**)  
 **else**:  
 **for** y **in** range(2,n):  
 **for** i **in** range(2,y):  
 **if** y%i == 0 :  
 *#print(n , " not Prime Number")* **break  
 else** :  
 **print**(y)  
  
prime(29)

# Python program to print all

# prime number in an interval

start = 11

end = 25

for val in range(start, end + 1):

   # If num is divisible by any number

   # between 2 and val, it is not prime

   if val > 1:

       for n in range(2, val):

           if (val % n) == 0:

               break

       else:

           print(val)

------------------

### **Write a program in Python to check if a sequence is a Palindrome.**

**def** palindrome\_str(s):  
 **if** s == s[::-1]:  
 **print**(**"palin"**)  
 **else** :  
 **print**(**'not'**)  
palindrome\_str(**'324'**)

----------------------------------

### **Q70. Write a one-liner that will count the number of capital letters in a file. Your code should work even if the file is too big to fit in memory.**

**Ans:**  Let us first write a multiple line solution and then convert it to one-liner code.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | with open(SOME\_LARGE\_FILE) as fh:  count = 0  text = fh.read()  for character in text:      if character.isupper():  count += 1 |

We will now try to transform this into a single line.

|  |  |
| --- | --- |
| 1 | count sum(1 for line in fh for character in line if character.isupper()) |

**def** cap\_str(s):  
 c = 0  
 **for** i **in** s:  
 **if** i == i.capitalize() :  
 c = c+1  
 **print**(**'count of capital letters is'**,c)  
  
cap\_str(**'AcbdAS'**)

### **Q85. What is map function in Python?**

**Ans:** map function executes the function given as the first argument on all the elements of the iterable given as the second argument. If the function given takes in more than 1 arguments, then many iterables are given. #Follow the link to know more similar functions.

The **map()** function in Python has two parameters, function and iterable. The map() function takes a function as an argument and then applies that function to all the elements of an iterable, passed to it as another argument. It returns an object list of results.

For example:

def calculateSq(n):

return n\*n

numbers = (2, 3, 4, 5)

result = map( calculateSq, numbers)

print(result)

### **Q86. Is python numpy better than lists?**

**Ans:** We use python numpy array instead of a list because of the below three reasons:

1. Less Memory
2. Fast
3. Convenient

For more information on these parameters, you can refer to this section – [Numpy Vs List](https://www.edureka.co/blog/python-numpy-tutorial/#NumpyVsList).

## **Python NumPy Array v/s List**

We use python numpy array instead of a list because of the below three reasons:

1. Less Memory
2. Fast
3. Convenient

The very first reason to choose python numpy array is that it occupies less memory as compared to list. Then, it is pretty fast in terms of execution and at the same time it is very convenient to work with numpy. So these are the major advantages that python numpy array has over list. Don’t worry, I am going to prove the above points one by one practically in PyCharm. Consider the below example:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | import numpy as np    import time  import sys  S= range(1000)  print(sys.getsizeof(5)\*len(S))    D= np.arange(1000)  print(D.size\*D.itemsize) |

O/P –  14000

4000

The above output shows that the memory allocated by list (denoted by S) is 14000 whereas the memory allocated by the numpy array is just 4000. From this, you can conclude that there is a major difference between the two and this makes python numpy array as the preferred choice over list.

Next, let’s talk how python numpy array is faster and more convenient when compared to list.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | import time  import sys    SIZE = 1000000    L1= range(SIZE)  L2= range(SIZE)  A1= np.arange(SIZE)  A2=np.arange(SIZE)    start= time.time()  result=[(x,y) for x,y in zip(L1,L2)]  print((time.time()-start)\*1000)    start=time.time()  result= A1+A2  print((time.time()-start)\*1000) |

O/P – 380.9998035430908  
49.99995231628418

### **Q87. How to get indices of N maximum values in a NumPy array?**

**Ans:** We can get the indices of N maximum values in a NumPy array using the below code:

|  |  |
| --- | --- |
| 1  2  3 | import numpy as np  arr = np.array([1, 3, 2, 4, 5])  print(arr.argsort()[-3:][::-1]) |

Output

[ 4 3 1 ]

What does this stuff mean: \*args, \*\*kwargs? And why would we use it?

### Answer

Use \*args when we aren't sure how many arguments are going to be passed to a function, or if we want to pass a stored list or tuple of arguments to a function. \*\*kwargs is used when we dont know how many keyword arguments will be passed to a function, or it can be used to pass the values of a dictionary as keyword arguments. The identifiers args and kwargs are a convention, you could also use \*bob and \*\*billy but that would not be wise.

What do these mean to you: @classmethod, @staticmethod, @property?

### Answer Background Knowledge

These are decorators. A decorator is a special kind of function that either takes a function and returns a function, or takes a class and returns a class. The @ symbol is just syntactic sugar that allows you to decorate something in a way that's easy to read.

Python decorator is a relative change that you do in Python syntax to adjust the functions quickly.

**@my\_decorator**

**def** **my\_func**(stuff):

do\_things

Is equivalent to

**def** **my\_func**(stuff):

do\_things

my\_func = my\_decorator(my\_func)

Describe Python's garbage collection mechanism in brief.

### Answer

A lot can be said here. There are a few main points that you should mention:

* Python maintains a count of the number of references to each object in memory. If a reference count goes to zero then the associated object is no longer live and the memory allocated to that object can be freed up for something else
* occasionally things called "reference cycles" happen. The garbage collector periodically looks for these and cleans them up. An example would be if you have two objects o1 and o2 such that o1.x == o2 and o2.x == o1. If o1 and o2 are not referenced by anything else then they shouldn't be live. But each of them has a reference count of 1.
* Certain heuristics are used to speed up garbage collection. For example, recently created objects are more likely to be dead. As objects are created, the garbage collector assigns them to generations. Each object gets one generation, and younger generations are dealt with first.

This explanation is CPython specific.

Place the following functions below in order of their efficiency. They all take in a list of numbers between 0 and 1. The list can be quite long. An example input list would be [random.random() for i in range(100000)]. How would you prove that your answer is correct?

**def** **f1**(lIn):

l1 = sorted(lIn)

l2 = [i **for** i **in** l1 **if** i<0.5]

**return** [i\*i **for** i **in** l2]

**def** **f2**(lIn):

l1 = [i **for** i **in** lIn **if** i<0.5]

l2 = sorted(l1)

**return** [i\*i **for** i **in** l2]

**def** **f3**(lIn):

l1 = [i\*i **for** i **in** lIn]

l2 = sorted(l1)

**return** [i **for** i **in** l1 **if** i<(0.5\*0.5)]

### Answer

Most to least efficient: f2, f1, f3. To prove that this is the case, you would want to profile your code. Python has a lovely [profiling package](https://docs.python.org/2/library/profile.html) that should do the trick.

**import** cProfile

lIn = [random.random() **for** i **in** range(100000)]

cProfile.run('f1(lIn)')

cProfile.run('f2(lIn)')

cProfile.run('f3(lIn)')

For completion's sake, here is what the above profile outputs:

**>>>** cProfile.run('f1(lIn)')

4 function calls **in** 0.045 seconds

Ordered by: standard name

ncalls tottime percall cumtime percall filename:lineno(function)

1 0.009 0.009 0.044 0.044 <stdin>:1(f1)

1 0.001 0.001 0.045 0.045 <string>:1(<module>)

1 0.000 0.000 0.000 0.000 {method 'disable' of '\_lsprof.Profiler' objects}

1 0.035 0.035 0.035 0.035 {sorted}

**>>>** cProfile.run('f2(lIn)')

4 function calls **in** 0.024 seconds

Ordered by: standard name

ncalls tottime percall cumtime percall filename:lineno(function)

1 0.008 0.008 0.023 0.023 <stdin>:1(f2)

1 0.001 0.001 0.024 0.024 <string>:1(<module>)

1 0.000 0.000 0.000 0.000 {method 'disable' of '\_lsprof.Profiler' objects}

1 0.016 0.016 0.016 0.016 {sorted}

**>>>** cProfile.run('f3(lIn)')

4 function calls **in** 0.055 seconds

Ordered by: standard name

ncalls tottime percall cumtime percall filename:lineno(function)

1 0.016 0.016 0.054 0.054 <stdin>:1(f3)

1 0.001 0.001 0.055 0.055 <string>:1(<module>)

1 0.000 0.000 0.000 0.000 {method 'disable' of '\_lsprof.Profiler' objects}

1 0.038 0.038 0.038 0.038 {sorted}

### **Q88. How do you calculate percentiles with Python/ NumPy?**

**Ans:** We can calculate percentiles with the following code

|  |  |
| --- | --- |
| 1  2  3  4 | import numpy as np  a = np.array([1,2,3,4,5])  p = np.percentile(a, 50) #Returns 50th percentile, e.g. median  print(p) |

Output

3

### **Q89**

Separate uniqies and duplicates

**def** uniq\_dup():  
 s = [1, 2, 2, 3, 4, 6, 2, 4, 6, 1, 7]  
 uniq = []  
 dup = []  
 **for** i **in** s:  
 **if** i **in** uniq :  
 dup.append(i)  
 **else** :  
 uniq.append(i)  
 **print**(uniq)  
 **print**(dup)  
  
uniq\_dup()

With enumerators, it’s easy to find an index while you’re inside a loop.

testlist = [10, 20, 30]

for i, value in enumerate(testlist):

print(i, ': ', value)

#1-> 0 : 10

#2-> 1 : 20

#3-> 2 : 30

We can use the following approach to create enum definitions.

class Shapes:

Circle, Square, Triangle, Quadrangle = range(4)

print(Shapes.Circle)

print(Shapes.Square)

print(Shapes.Triangle)

print(Shapes.Quadrangle)

#1-> 0

#2-> 1

#3-> 2

#4-> 3

To find the version of python code

**import** sys  
  
**print**(sys.version)

### **Use \_\_slots\_\_ To Reduce Memory Overheads.**

Have you ever observed your Python application consuming a lot of resources especially memory? Here is one trick which uses <\_\_slots\_\_> class variable to reduce memory overhead to some extent.

import sys

class FileSystem(object):

def \_\_init\_\_(self, files, folders, devices):

self.files = files

self.folders = folders

self.devices = devices

print(sys.getsizeof( FileSystem ))

class FileSystem1(object):

\_\_slots\_\_ = ['files', 'folders', 'devices']

def \_\_init\_\_(self, files, folders, devices):

self.files = files

self.folders = folders

self.devices = devices

print(sys.getsizeof( FileSystem1 ))

#In Python 3.5

#1-> 1016

#2-> 888

Clearly, you can see from the results that there are savings in memory usage. But you should use \_\_slots\_\_ when the memory overhead of a class is unnecessarily large. Do it only after profiling the application. Otherwise, you’ll make the code difficult to change and with no real benefit.

n = 18  
**print**(**'Ravi {}'**.format(n) )

#Ravi 18

g = raw\_input(**'Enter :'**)  
**print** g

input for string

# Python 2.7.6

>>> a = raw\_input("enter :- ")

enter :- 3

>>> type(a) # raw\_input() converts your int to string

<type 'str'>

>>> a = input("enter :- ")

enter :- 3

>>> type(a) # input() preserves the original type, no conversion

<type 'int'>

>>>

>>> a = input("enter name :- ")

enter name :- 'Derrick'

>>> a = '123456'

>>> print a.split('2', 1)[0]

1

>>> print a.split('4', 1)[0]

123

>>>

### **Q-10: What Are The Principal Differences Between The Lambda And Def?**

#### **Lambda Vs. Def.**

* Def can hold multiple expressions while lambda is a uni-expression function.
* Def generates a function and designates a name to call it later. Lambda forms a function object and returns it.
* Def can have a return statement. Lambda can’t have return statements.
* Lambda supports to get used inside a list and dictionary.

### **Q-70: What Are Closures In Python?**

Python closures are function objects returned by another function. We use them to eliminate code redundancy.

In the example below, we’ve written a simple closure for multiplying numbers.

def multiply\_number(num):

def product(number):

'product() here is a closure'

return num \* number

return product

num\_2 = multiply\_number(2)

print(num\_2(11))

print(num\_2(24))

num\_6 = multiply\_number(6)

print(num\_6(1))

The output is:

22

48

6

### **Q-84: Why Do You Use The Zip() Method In Python?**

The zip method lets us map the corresponding index of multiple containers so that we can use them using as a single unit.

Signature:

zip(\*iterators)

Arguments:

Python iterables or collections (e.g., list, string, etc.)

Returns:

A single iterator object with combined mapped values

# Example: zip() function

emp = [ "tom", "john", "jerry", "jake" ]

age = [ 32, 28, 33, 44 ]

dept = [ 'HR', 'Accounts', 'R&D', 'IT' ]

# call zip() to map values

out = zip(emp, age, dept)

# convert all values for printing them as set

out = set(out)

# Displaying the final values

print ("The output of zip() is : ",end="")

print (out)

The output is:

The output of zip() is : {('jerry', 33, 'R&D'), ('jake', 44, 'IT'), ('john', 28, 'Accounts'), ('tom', 32, 'HR')}

Table below explains the difference between Python version 2 and Python version 3.

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No** | **Section** | **Python Version2** | **Python Version3** |
| 1. | Print Function | Print command can be used without parentheses. | Python 3 needs parentheses to print any string. It will raise error without parentheses. |
| 2. | Unicode | ASCII str() types and separate Unicode() but there is no byte type code in Python 2. | Unicode (utf-8) and it has two byte classes −   * Byte * Bytearray S. |
| 3. | Exceptions | Python 2 accepts both new and old notations of syntax. | Python 3 raises a SyntaxError in turn when we don’t enclose the exception argument in parentheses. |
| 4. | Comparing Unorderable | It does not raise any error. | It raises ‘TypeError’ as warning if we try to compare unorderable types. |

### **26. How are Python arrays and Python lists different from each other?**

In Python, when we say ‘arrays’, we are usually referring to ‘lists’. It is because lists are fundamental to Python just as arrays are fundamental to most of the low-level languages.

However, there is indeed a module named ‘array’ in Python, which is used or mentioned very rarely. Following are some of the differences between Python arrays and Python lists.

|  |  |
| --- | --- |
| **Arrays** | **Lists** |
| Arrays can only store homogeneous data (data of the same type). | Lists can store heterogeneous and arbitrary data. |
| Since only one type of data can be stored, arrays use memory for only one type of objects. Thus, mostly, arrays use lesser memory than lists. | Lists can store data of multiple data types and thus require more memory than arrays. |
| The length of an array is pre-fixed while creating it, so more elements cannot be added. | Since the length of a list is not fixed, appending items to it is possible. |

### **39. What is the difference between append() and extend() methods?**

Both append() and extend() methods are methods used to add elements at the end of a list.

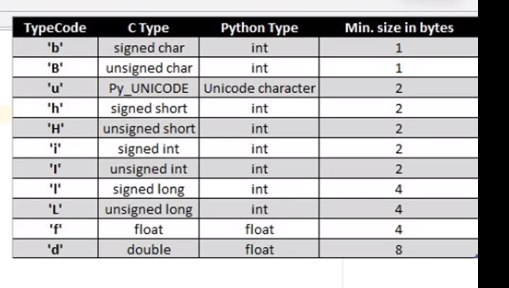
* **append(element)**: Adds the given element at the end of the list that called this append() method
* **extend(another-list)**: Adds the elements of another list at the end of the list that called this extend() method
* **reduce()**: Reduce repeatedly reduces a sequence pair-wise until it reaches a single value.

>>> from functools import reduce >>> reduce(lambda x,y:x-y,[1,2,3,4,5]) -13

#### 7. Check the memory usage of  an object

|  |  |
| --- | --- |
| 1  2  3 | import sys  x = 10  print(sys.getsizeof(x)) |

**def** sqrt(n):  
 **print**(round((n\*\*0.5),2))  
  
sqrt(4)



2-d array

arr = []  
  
**for** arr\_i **in** range(6):  
 arr\_t = [int(arr\_temp) **for** arr\_temp **in** raw\_input().strip().split()]  
 arr.append(arr\_t)  
total = []  
  
**for** i **in** range(len(arr)-2):  
 **for** j **in** range(len(arr)-2):  
 total1 = sum(arr[i][j:j + 3]) + arr[i + 1][j + 1] + sum(arr[i + 2][j:j + 3])  
 *#print(total1)* total.append(total1)  
 **print**(total)  
  
**print**(max(total))

The [print function](https://docs.python.org/3/library/functions.html#print) uses sep to separate the arguments, and end after the last argument. Your example was confusing because you only gave it one argument. This example might be clearer:

>>> print('boa', 'cat', 'dog', sep=', ', end='!!!\n')

boa, cat, dog!!!

Of course, sep and end only work in Python 3's print function. For Python 2, the following is equivalent.

>>> print ', '.join(['boa', 'cat', 'dog']) + '!!!'

boa, cat, dog!!!

**def** array\_left\_rotation(a, n, k):  
 **return** a[k:]+a[:k]  
  
n, k = map(int, raw\_input().strip().split(**' '**))  
a = list(map(int, raw\_input().strip().split(**' '**)))  
answer = array\_left\_rotation(a, n, k)  
  
answer = [str(i) **for** i **in** answer]  
  
**print**(answer[0:3])

d = {v:i **for** i,v **in** enumerate(a)}

print 5/3 = 1

print 5//3 = 1

print 5.0/3.0 = 1.667

print 5.0//3.0 = 1.0

<https://www.ics.uci.edu/~pattis/ICS-33/lectures/complexitypython.txt>

<http://www.learn4master.com/interview-questions/leetcode/leetcode-problems-classified-by-company>

<https://www.inventwithcode.com/crushing-coding-interviews/>

float("inf")

imp = sorted([ x for x, y in contests if y==1 ], reverse=True)

Abstract Class – A class which does not allow instantiation of object.

<https://www.youtube.com/watch?v=6Fs2MIf61oQ>

\_\_init\_\_ method

<https://www.edx.org/>

thenewboston

telusko

*super(Child, self).\_\_init\_\_()*

def computeDifference(self):

self.maximumDifference = max([abs(x1 - x2)] for x1 in self.\_\_elements for x2 in self.\_\_elements)[0]

## **Example**

>>> class MyClass():

... def \_\_init\_\_(self):

... self.\_\_superprivate = "Hello"

... self.\_semiprivate = ", world!"

...

>>> mc = MyClass()

>>> print mc.\_\_superprivate

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

AttributeError: myClass instance has no attribute '\_\_superprivate'

>>> print mc.\_semiprivate

, world!

>>> print mc.\_\_dict\_\_

{'\_MyClass\_\_superprivate': 'Hello', '\_semiprivate': ', world!'}

<https://www.codefellows.org/courses/code-401/advanced-software-development-in-python/>